Seminar "'Open Source Software Engineering"´

Winterterm 2004

Tools for light weight knowledte sharing in open-source software development

Maximilian Höflich

hoeflich@inf.fu-berlin.de

Advisor: Christopher Oezbek

02.09.2005

Contents

1 Einleitung

Wissensmanagement in "Free and Open Source"(FOSS) Projekten ist allgemein gesagt das Thema. Ausgehend von einer Definition von Wissensmanagement und der Definition ihrer Aufgaben und Probleme wird aufgezählt mit welchen Mitteln Wissen vermittelt wird. Hierbei bleibt der Fokus auf FOSS Projekten und ihren spezifischen Problemstellungen.

Die zu lösende Problemstellung ist die eines Programmierers, der einem Projekt neu beitritt. Er steht vor einer meist sehr fortgeschrittenen Anwendungen, an der er mitschreiben will.

Selbst wenn eine strukturierte API (Application Programming Interface) Dokumentation existiert, so ist diese bei grösseren Projekten meist so umfangreich und oder komplex, das durch Lesen allein nicht immer zu einem Resultat gefunden werden kann. Hier sind Tipps oder Erklärungen nützlich, wie sie z.B. der Mitarbeiter am nächsten Tisch zur Verfügung hat. Leider hat der durchschnittliche FOSS Programmierer diesen nicht, weil er allein arbeitet.

Es kann aber auch sein, das gar keine, nur wenige oder unvollständige Dokumentation existiert. "Light weight knowledge sharing" ist dabei ein Ansatz, der auf dem technischen "Ist"-Stand, also den üblicherweise verwendeten Tools und Techniken aufsetzt und diese neu strukturiert und somit leichter les- und analysierbar macht. Desweiteren versucht dieser Ansatz auch das Fehlen einer bestimmten Informationsquelle (z.B. eine fehlende API-Dokumentation) durch das Verknüpfen anderer Quellen auszugleichen.

2 Wissensmanagement

Wissensmanagement beschreibt die Organisation der Wissensbasis eines Unternemehmens, mit dem Ziel dieses zum optimalen Nutzen einzusetzen.

Die Wissensbasis [???a] eines Unternehmens umfasst alle Informationen, Daten sowie das Wissen der Mitarbeiter. Wissen ist hierbei immer an Personen gebunden und beschreibt die Gesamtheit der Kenntnisse und Fähigkeiten um Probleme zu lösen. Es wird unterteilt in explizites und implizites Wissen.

Explizites Wissen beschreibt hierbei sogenanntes Aussagewissen, das leicht durch Schrift und Zeichen repräsentierbar ist. Im Gegensatz dazu steht implizites Wissen, welches auch Erfahrungswissen genannt wird. Dieses beschreibt die Fähigkeit Entscheidungen aufgrund von Wahrnehmung, Beurteilung und Analyse zu treffen. Es ist weitaus schwerer zu erlernen und zu archivieren [???d].

2.1 Das Münchner Modell

Wissensmanagement kann als Kreislauf beschrieben werden [???b]. Abbildung 1 zeigt, wie ausgehend von einer Zielsetzung Wissensmanagement über vier zentrale Prozesse erfolgt, deren Ergebnisse evaluiert werden und eventuell eine veränderte Zielsetzung zur Folge haben.

Wissensrepräsentation beschreibt den Prozess der Identifikation und Dokumentation von Wissen. In der Wissenskommunikation geht es darum das gut strukturierte Wissen zu verteilen. Der Austausch und die Vermittlung stehen hierbei im Vordergrund.

Wissensgenerierung beschäftigt sich mit der Verarbeitung von bestehendem Wissen, sowie mit dem Ziel neues zu entwickeln und aufzubauen, z.B. durch Hinzufügen (kaufen) von externem Wissen oder durch Forschung. Was nutzen die vorhergegangenen drei Prozesse, wenn niemand das Wissen nutzt?

Die Wissensnutzung hat zur Aufgabe Wissen in Produkte und Dienstleistungen umzusetzen. Es ist ein Mass für die Effektivität des gesamten Wissensmanagements.

Abbildung 1 aus [???b]

2.2 Probleme [???c]

Kollektivierung des Wissens:

Oftmals ist die Motivation Wissen weiterzugeben nicht sehr gross. Das Sprichwort ”Wissen ist Macht” bewahrbeitet sich in vielen Unternehmen, wenn z.B. ein individueller Wissensvorsprung die Machtposition verbessert.

Dieses Problem besteht in FOSS Projekten eher selten.

Institutionalisierung des Wissen:

Dieses Problem spricht folgende Fragen der Form der Wissensspeicherung an:

Wird spezifisches Wissen schnell und zuverlässig gefunden ?

Sind die Informationen in einer leicht verständlichen Form ?

Sind die gefundenen Informationen ausreichend ?

Akzeptanzproblem:

Die gewählte Art der Repräsentation des Wissens und die damit verbundenen Strukturen, Regeln und Handlungsweisen müssen von den Mitarbeitern akzeptiert und angewandt werden.

Damit geht oftmals eine Ãnderung der Unternehmenskultur einher.

Rückkopplungsproblem:

Im optimalen Fall stösst Wissensmanagement einen zyklischen Prozess an. Das Rückkopplungsproblem thematisiert inwieweit die Wissensgenerierung durch das eigene Wissensmanagement unterstützt wird. Gelingt es dem Unternehmen diesen Kreislauf in Gang zu setzen und zu halten?

2.3 Parameter in traditionellen Organisationen

In traditionellen Unternehmen gibt es für die Mitarbeiter vor allem zwei grosse Motivationen ihren Job zu machen: Geld und Karriere.

Die Kollektivierung des Wissens ist nicht lohnenswert, denn der Karriere der Einzelperson ist ein persönlicher Wissensvorsprung viel zuträglicher.

Viele Unternehmen versuchen das Akzeptanzproblem zu lösen, indem sie finanzielle Anreize nutzen [???a].) Die Institutionalisierung des Wissens wird zum Problem, wenn z.B. alteingesessene Mitarbeiter, die schon seit Ewigkeiten nach Methode A gehandelt haben nun Methode B benutzen sollen und dazu nicht bereit sind.

Wissensmanagement in traditionellen Betrieben kann geordnete Bahnen gehen.

Ein Unternehmen hat mehrere Niederlassungen. Wird ein Mitarbeiter neu eingestellt, dann ist er einer Niederlassung (mehr oder weniger) fest zugeordnet.

Neben einer wie auch immer gearteten Dokumentation wird ihm für die Anfangszeit meist ein Mentor zugeteilt, der ihn in das zu bearbeitende Projekt einführt. Dieser Mentor hat bereits Erfahrungen und kann Unklarheiten im Gespräch beseitigen.

Dieses Mentoring ist essentieller Bestandteil und durchzieht die Kommunikationskultur in traditionellen Unternehmen.

2.4 Parameter in FOSS Projekten

In FOSS Projekten ist der einzelne Mitarbeiter aus persönlichem Interesse Teil des Projektes. Seine Motivation ist oftmals der Wissenszuwachs.

Die Kollektivierung des Wissens ist hier kein Problem, da alle Informationen über ein Projekt für jeden z.B. in Form des Quelltextes ersichtlich sind. Ein FOSS Projekt steht jedem offen, der Interesse hat sich einzubringen. Niemand verlangt dabei sofort als Kernmitglied aufgenommen zu werden, aber über Patches und andere Beiträge kann mitgearbeitet werden. Da dieses auch im Interesse der Entwickler ist, werden Dokumentationen ausnahmslos der Öffentlichkeit bereitgestellt.

Ein Zurückhalten von Informationen bringt keinem etwas, da das Individuum keinen persönlichen Vorsprung gegenüber seinen Mitstreitern erlangt.

Die Probleme der Institutionalisierung und der Akzeptanz sind wie in traditionellen Organisationen vorhanden.

Die meisten FOSS Projekte werden jedoch als Community Projekte organisiert [Kev03].Die Fluktuation an Mitarbeitern ist hierbei sehr gross und dementsprechend hoch ist auch das Bewusstsein für die Notwendigkeit von Dokumentationen. Dies gilt natürlich nicht für Projekte, die nur aus einer geringen Anzahl Programmierern bestehen, sondern bezieht sich vornehmlich auf grosse Projekte, wie z.B. das KDE Projekt oder Eclipse ...

All das hört sich so an, als ob FOSS Projekte vorbildlich dokumentiert wären.

Wie sieht also Wissensmanagement in FOSS Projekten aus und wo liegen die Probleme?

FOSS Projekte beginnen meist mit einer kleinen Anzahl von Programmierern und einer Idee. Ist die Idee gut, dann motiviert sie vielleicht andere, die sich entschliessen ihre Arbeitskraft zur Verfügung zu stellen.

Hier das Beispiel KDE:

KDE [1] steht für Kool Desktop Enviroment. Im Oktober 1996 fanden sich 3 Entwickler zusammen und entdeckten Qt [2] für sich. Qt ist ein Widget Framework, ähnlich der Swing, das plattformübergreifende Entwicklung von GUI Applikationen erlaubt.

Auf Basis dieser Bibliothek sollte eine Desktop Umgebung programmiert werden, die sich an CDE (Common Desctop Enviroment) orientiert, welches unter einer proprietären Lizenz vertrieben wurde und wird.

Betrachtet man das KDE Projekt heutzutage, so sind aus den 3 Entwicklern allein im Bereich des Kerns ca. 20 geworden [3]. Daneben existieren in den ca. 40 Nebenprojekten mehrere Hundert aktive Entwickler.

Dieser Zuwachs an Arbeitskraft geht jedoch nicht geordnet vonstatten und niemand kann sagen aus welchem Land, oder von welchem Kontinent diese Person kommt.

Es ist möglich, dass die Programmierer niemals Gelegenheit haben sich zu treffen oder persönlich ihre Erfahrungen auszutauschen. Diese Verteilung bringt noch das weitere Problem der Zeitzonen mit sich. Selbst wenn Person A vielleicht gerade erreichbar ist, kann es sein, das Person B schon/noch/gerade schläft.

Obwohl diese Barrieren überwunden werden können und man sich zu einer für beide annehmbaren Zeit online verabredet hat, ist noch nicht gesagt, dass sie sich auch verstehen. Englisch ist zwar in den meisten Projekten die Sprache zur Kommunikation, aber dass diese gut beherrscht wird ist nicht sicher.

Diese Probleme betreffen vor allem synchrone Informationsmedien, weshalb in FOSS Projekten die asynchrone Wissensvermittlung dominiert.

---

[1] http://www.kde.org

[2] http://www.trolltech.com

[3] http://sunsite.bilkent.edu.tr/pub/linux/www.kde.org/gallery/

3 Informationen

Informationen sind nicht gleich Informationen.

Ich betrachte im folgenden das Ausgangsszenario, in dem ein Programmierer einem bestehenden weit fortgeschrittenem Projekt beitritt.

Dieser sieht sich einem grossen Wissensdefizit gegenüber, das beseitigt werden muss durch :

- Dokumentationen für Benutzer (was macht die Software)
- Dokumentationen für Entwickler (wie macht es die Software)

Die Informationen als Entwickler sind im einzelnen :

- Quellbeispiele
- strukturierte Tutorials
- Codingstandards / andere Vereinbarungen
- Koordination des gesamten Projekts

Betrachtet man nun bestehende FOSS Projekte und analysiert deren Informationsfluss, so können 2 unterschiedliche Arten von Informationsmedien identifiziert werden: synchrone und asynchrone.

3.1 synchrone Informationsmedien

Synchrone Informationsmedien zeichnen sich dadurch ab, dass sie dialogorientiert in Echtzeit zwischen realen Personen ablaufen. Besonders in traditionellen Organisationen wird Wissensvermittlung hauptsächlich synchron betrieben. [Tho04a] Eine Person, die einem Projekt betritt bekommt eine MentorIn an die Seite gestellt. Dieser steht physikalisch bereit und kann direkte Hilfestellungen geben.

Diese Art der Kommunikation wird als Mentoring oder Support beschrieben. Leider wird heutzutage ein schnödes FAQ schon teilweise als ausreichender "Support" betrachtet, weshalb Mentoring der treffendere Begriff ist.

Die im folgenden Vorgestellten Werkzeuge eignen sich vor allem für die richtigen Hinweise. "Wenn du wissen willst, wie dieser Teil des Frameworks zusammenarbeitet, dann guck dir URL ... an".

3.1.1 Instant Messanger /ICQ, AOL,...) z.B. Gaim, Kopete

Probleme: verschiedene Zeitzonen, verschiedene Muttersprachen, Chats mit mehreren Personen nur umständlich möglich.

Vorteile: Erreichbarkeit ist höher, als bei traditionellem Mentoring, das weghören jedoch auch.

3.1.2 GnomeMeeting, NetMeeting (Audio/Video)

Probleme: wie bei Instant Messengers, plus technologische Hürde (Webcam, Soundkarte, Microphon), desweiteren muss für einen Audiocodec mit annehmbarer Qualität auch eine entsprechende Bandbreite vorhanden sein.

Problematisch ist auch die Fähigkeit Englisch zu sprechen in internationalen Teams.

Vorteile: Mentoring in ähnlicher Qualität, wie "live-Mentoring", ausserdem werden auch Metainformationen übertragen, wie Gesichtsausdrücke und Stimmlagen, was z.B. die Schwere eines Problems besser einschätzen lässt.

3.1.3 IRC

Internet Relay Chat ist ein in Entwicklerkreisen inzwischen halbwegs anerkanntes Medium. Seine Probleme sind ähnlich denen des Instant Messenger, jedoch ist er ausgelegt auf Chats mit vielen Personen.

Leider sind die Kommunikationsstränge nicht geordnet, so das zu belebten Zeiten mehrere Gruppen im selben Chat über andere Themen reden können. Das macht es für Tools schwer auswertbar. Desweiteren sind viele triviale Themen dabei.

Vorteile: leichteste Benutzung aller Synchronen Informationsmedien. Ein IRC-client st in jeder Linux-Distribution dabei und auch unter Windows kostenlos erhältlich.

3.1.4 Probleme Synchroner Informationsmedien

Das grösste Problem des Mentoring ist jedoch, das keines der vorgestellten Tools eine gute Möglichkeit der Archivierung bietet. Informationen, müssen für jeden Wissenstransfer wiederholt repliziert werden.

Desweiteren ist Geschwindigkeit der Wissensübertragung langsam. In einem Gespräch kann sehr schnell herausgefunden werden, was das eigentliche Problem ist, um dann ohne Umschweife zu einem Lösungsansatz zu gelangen. Das getippte Wort ist hier um einiges ungenauer, da z.B. nur beschränkt Metakommunikation existiert.

3.2 asynchrone Informationsmedien

Asynchrone Informationsmedien sind die traditionellen Informationsspeicher. Unter diese Kategorie fallen alle Arten von koordiniert gespeicherten und asynchron gepflegten Informationsbestände.

Diese Art der Informationsmedien wird gleichermassen in traditionellen, wie auch in FOSS Organisationen benutzt.

FOSS Projekte sind jedoch Informationstechnisch eher lose organisiert. Es gibt kaum jemanden, der hauptsächlich dafür zuständig ist die Position eines Wissensmanagers einzunehmen. Daher werden hauptsächlich allgemein bekannte Tools benutzt, die allein gesehen gut als Wissensspeicher funktionieren, aber weitergehende Funktionen oftmals vermissen lassen.

Diese Tools werden desweiteren meist ausgesucht, weil eine Person im Team die Möglichkeit hat sie einzustzen und bereit ist die Administration zu übernehmen.

Ist ein Informationsmedium in einem Projekt eingeführt, sagt dies nichts über die Qualität der Dokumentation oder die Bedienbarkeit aus. Ein Teil der Programmierer kann sehr viel dokumentieren, während ein anderer dies nicht tut.

In FOSS Projekten gibt es im Gegensatz zu traditionellen Organisationen kaum eine Möglichkeit einen überzeugten Dokumentationsverweigerer dazu zu bringen dies doch zu tun, solange dieser nicht Angestellter einer Firma ist.

Der unberechenbare Faktor Mensch bleibt.

3.2.1 Wissensportale [Dav03]

Wissensportale werden eher in traditionellen Organisationen eingesetzt, die einen Anreiz über meist finanzielle Wege schaffen, so dass diese von ihren Mitarbeitern benutzt und erweitert werden.

Wissensportale, wie "knowledgemotion" [???c] bestehen hauptsächlich aus einem grossen CMS und einer Groupwarelösung. Diese zusammengenommen ermöglichen duch Autorisierung gegenüber dem System einen nutzerspezifisch angezeigten Informationsbestand.

Diese Wissensportale benötigen eigene Rechner auf denen sie laufen und Administratoren, die sich nur um die technische Administration der Systeme kümmern. Dies ist ein Aufwand, den sich die meisten FOSS Projekte nicht leisten können.

Abgesehen von finanziellen Aspekten gibt es kein Wissensportal, das sich als "Common Practice", also allgemein Anerkannt in der FOSS Community durchgesetzt hat.

3.2.2 statische Homepage

Häufig existiert für ein Projekt nur eine statische Homepage

Vorteil: Administration ist in einer Hand, Strukturierung damit meist klar

Nachteil: viel Arbeit für eine Person, Homepage meist veraltet, Potential liegt brach

3.2.3 CMS Homepage [Tho04a]

Vorteil: alle mit Zugang können Content beitragen

Nachteil: Strukturelle Änderungen immer noch nur von wenigen "HeadAdmins"

Suchfunktionalität beschränkt sich auf Wortvergleich.

Unter den CMS gibt es "eierlegende Wollmilchsäue" mit allen möglichen Zusatzfunktionen für Projektmanagement und teilweise auch Wissensmanagement , wie z.B. Zope [4].

[4] http://www.zope.org

3.2.4 Wiki [Tho04a]

Vorteile: Alle können editieren und die Struktur ändern

Nachteile: Suchfunktionalität beschränkt sich auf Wortvergleich. Alle können editieren (Abbildung 2 vom 08.07.05)

z.B. JSP Wiki [5] bietet einem eine Plugin Schnittstelle, auf Basis derer schon viele Plugins realisiert wurden.

Unter diese fallen auch EB und MASE [Tho] [Tho04a], welche Unterstützung für knowledge sharing und Prozessorganisation bieten.

Abbildung 2

3.2.5 Mails

Mails sind wohl das älteste Medium, um Wissen über das Internet zu vermitteln.

Vorteile: Dialoge mit realen Personen ermöglichen gezielte Fragestellungen

Nachteile: Nicht so nah wie Mentoring, Nur lokal archiviert (personengebunden) und nicht öffentlich zugänglich, wenig strukturiert (schwer nach Informationen zu durchsuchen)

3.2.6 Mailinglisten/Newsgroups

Nachteile: siehe Mails

Vorteile: öffentlich oder teilöffentlich einsehbar, archiviert und durchsuchbar

3.2.7 Sourcedokumentationsn (z.B. JavaDoc)

Eigentlich kein eigenes Medium, sondern als Ausgabe eingebettet in eine statische oder dynamische Homepage bieten Sourcedokumentation Beschreibungen der Funktionalitaet eines Projektes.

Dies geschieht jedoch meist nur als Schnittstelle, so dass private Methoden nicht auftauchen (auch meist wenig dokumentiert).

Vorteile: Source und Dokumentation lokal beisammen

Nachteile: keine Suchfunktionalität, Informationen nur im kleinen Kontext erkennbar, Funktionalität eines Frameworks z.B. nicht einsehbar.

[5] http://www.jspwiki.org/

3.2.8 Bugzilla

Bugzilla ist grob gesagt ein Issue-System. Es können Aufträge eingetragen werden, die dann abgearbeitet werden.

Hauptsächlich wird dieses System benutzt, um Benutzern und Programmieren einer Software die Möglichkeit zu geben Fehler bekannt zu machen.

Es ist zwar nicht das klassisches Informationsmedium und beinhaltet eher Metainformationen, die nicht direkt Wissen vermitteln, sondern das Wissen wer es könnte. Z.B. kann davon ausgegangen werden, dass ein Programmierer, der in einem System einen Bug gefixed hat sich auch halbwegs darin auskennt.

3.3 Die versteckten Wissensspeicher

Von den vorgestellten synchronen und asynchronen Informationsmedien sind die Funktionen zur Informationsspeicherung und -verbreitung wohl bekannt.

Es existieren jedoch andere Medien, deren vordergründige Funktion auch das Bereitstellen von Wissen (mehr oder weniger vordergründig) ist, die aber weitere Informationen an Stellen besitzen, die man nicht erwartet.

3.3.1 CVS

Concurrent Versioning System oder kurz CVS bietet einem Entwicklerteam die Möglichkeit gleichzeitig an Dateien zu arbeiten, ohne ihre Änderungen gegenseitig zu überschreiben.

Die Funktionalität, die für uns besonders interssant ist, sind die "Commit Nachrichten". Wann immer Informationen in das CVS geschrieben werden (also eine Anzahl Quellen aktualisiert werden), wird dem System eine Nachricht mitgegeben.

Diese Nachrichten haben einen Datumsstempel und sind mit den geänderten Dateien verknüpft. Eine versteckte Information ist z.B. "Wann immer Klasse A verändert wurde, wird auch eine Änderung in Klasse B vorgenommen".

Diese Informationen macht sich z.B. das Tool "Rose" zunutze.

3.3.2 Bugzilla

Die versteckte Information ist bei Bugzilla in den BugZillaIDs zu finden, die z.B. in CVS Commit Nachrichten auftauchen können und hier signalisieren, das ein bestimmtes Issue damit abgearbeitet wurde.

4 Light weight knowledge sharing

Die im vorgehenden beschriebenen Werkzeuge werden in Teilen von FOSS Projekten zur Weitervermittlung von Wissen benutzt.

Sie haben alleingesehen ihre Vor- und Nachteile, eignen sich jedoch nie als alleinige Lösung. Es sind auch nie alle Werkzeuge gleichzeitig im Einsatz.

Abgesehen von den aufgezeigten Möglichkeiten, die einem die genannten Informationsspeicher bieten können diese die Mitarbeiter eines Projektes in der Dokumentation nur unterstützen.

Es gibt kein Tool, das Dokumentation automatisch schreibt und so ist der einzelne Entwickler für die Qualität dieser verantwortlich.

Das Problem der Akzeptanz führt in FOSS Projekten zu einer teilweise sehr durchwachsenen Dokumentation.

"Light Weight Knowledge Sharing" ist ein Ansatz, der bestehende Informationsquellen betrachtet und Zusammenhänge zwischen den Informationen herstellt um diese dann aufzuberiten und in einer besser strukturierten Form dem Nutzer bereitzustellen [Dav03]. Besonders wichtig ist hierbei die Möglichkeit diesen neu zusammengefügten Informationsstamm besser durchsuchen zu können.

Dabei wird davon ausgegangen, das es in FOSS Projekten "Common Practices" gibt. Diese sind in der Benutzung bestimmter Werkzeuge zu finden.

So wird davon ausgegangen, dass ein normales FOSS Projekt die folgenden benutzt:

- CVS/Subversion
- Bugzilla
- durchsuchbare Internetseiten
- Mailinglisten

Die Informationen aus diesen Quellen sind jedoch wenn überhaupt nur sehr lose gekoppelt.

Die Aufgabe von "Light Weight Knowledge Sharing Tools" ist es nun all diese Informationen miteinander zu verknüpfen, damit sie als Gesamtes verwertbar sind.

4.1 Hipikat [Dav]

Hipikat ist ein Werkzeug mit dessen Hilfe von einem Bugzilla Eintrag ausgehend andere Informationen, die damit in Verbindung stehen gefunden werden sollen.

Geht man davon aus, das ein Programmierer einen Aufgabe zugewiesen bekommen hat, die einem Task innerhalb Bugzillas entspricht, kann er sich mit Hilfe von Hipikat andere Bugzilla Tasks anzeigen lassen, die z.B. im gleichen Subsystem sind, oder überdeckende Klassen behandeln.

Die hierbei betrachteten Informationen kommen aus den Quellen: CVS, Bugzilla, Newsgroups und statischen Homepages.

Hipikat besteht aus einem Client-Server System. Auf dem Server werden die Informationen gesammelt und untereinander semantisch Verknüpft (Abbildung 3).

Das Resultat dieser Verknüpfung bildet ein "implizites Gruppengedächtnis", an das Anfragen gestellt werden können.

Man spricht von einem impliziten Gedächnis, da es nicht explizit erstellt werden muss, sondern aus vorhandenen Informationen generiert wird.

Zum gegenwärtigen Zeitpunkt war es dem Autor leider nicht möglich eine lauffähige Version des Werkzeugs zu erlangen.

4.1.1 Erstellen des "Gruppengedächntisses" / der Server

Die Verarbeitung der Informationen erfolgt über die Module: Update, Identification und Selection (Abbildung 4).

4.1.2 Update

Dieses Modul ist dafür zuständig regelmässig die Informationsquellen nach neuen Informationen zu durchsuchen. Hierbei wird das CVS nach neuen Updates befragt, die Newsgroups abgerufen, die Homepage textuell gescannt und Bugzillas Webschnittstellte nach neuen oder veränderten Einträgen abgesucht.

Es werden noch keine bearbeitenden/wertenden Schritte unternommen.

Abbildung 3

4.1.3 Identification

Das Updatemodul benachrichtigt das Identifikationsmodul, sobald es eine neue Information gefunden hat.

Anhand der Informationsquelle wird nun entschieden, wo diese dem Datenstamm hinzuzufügen ist (in welche Tabelle) und welche Relationen/Verknüpfungen hergestellt werden müssen. Für letzteres existieren die Submodule "log-matcher", "activity-matcher", "text similarity matcher" und "newsgroup thread matcher".

Handelt es sich bei der Informationsquelle um das CVS, so wird die Commitnachricht geholt und mit dem "log matcher" nach eventuell vorkommenden BugzillaIds durchsucht (z.B. "behebt Bug 32234"). Diese Information wird innerhalb der Datenbank als "implements" Verknüpfung zwischen einer "Change/Bug" Entität und einer "File revision" abgelegt, wobei innerhalb der "File revision" Tabelle nur die Metadaten (wie z.B. die Zeit in der der Eintrag ins CVS gemacht wurde) gespeichert werden, nicht jedoch der dazugehörige Quelltext.

Besteht die neue Information aus einem Bugzillaeintrag, so wird die Abarbeitung an den "activity-matcher" delegiert.

Hierbei wird von folgender Annahme ausgegangen: Ein Bugzilla Task wird lokal abgearbeitet und erst wenn er vollendet ist ins CVS geladen. Sobald dies passiert ist wird der Status des Bugzillaeintrages zeitlich sehr nah geändert (z.B. auf Closed). Da diese Änderungen nun sehr nah bei einanderliegen werden "File review" Entitäten, deren Zeitstempel nah an der Änderung dieses Eintrages liegen, in der Datenbank über die Verknüpfung "implements" verbunden.

Der "text similarity matcher" wird angewandt, wenn es sich bei den Informationen um eine Seite der Homepage handelt, eine CVS Commitnachricht, ein Kommentar innerhalb Bugzillas oder ein Newsgroupeintrag. Es stellt Verknüpfungen zwischen allen Entitäten innerhalb der Datenbank her.

Am einfachsten ist der "newsgroup thread matcher", welcher die Newsgroupeinträge nur nach Zusammengehörigkeit ordnet und die Überschriften nach Verweisen auf Bugzillaeinträge durchsucht.

Abbildung 4

4.1.1.3

4.1.4 Selection

Das Selectionmodul stellt die Anbindung für den Client dar und sorgt dafür, Anfragen mit sinnvollen Resultaten zu beantwortet.

Hierfür bedient es sich mehrer Submodule, die jeweils auf eine bestimmte Informationsquelle Spezialisert sind. Jedes einzelne erbringt Vorschläge, die dann zusammengemengt werden.

Eine Anfrage an den Server besteht immer aus einer bestimmten Entität in der Datenbank. Dies kann ein Bugzillaeintrag sein, oder eine Filerevision.

Realisiert wird dies über eine Webapplikation, die auf einem Tomcat Server läuft und ihre Webapplikation über Webservices anbietet (mittels SOAP).

4.1.5 eine Anfrage stellen / Der Client

Sobald Hipikat benutzt wird kann davon ausgegangen werden, das der Nutzer mit der entsprechenden Programmiersprache vertraut ist und sich einen generellen Überblick über das entsprechende Projekt verschafft hat.

Angenommen die Aufgabe eines Programmierers ist die Abarbeitung einer bestimmten in Bugzilla beschriebenen Aufgabe. Die ID des Eintrages kann als Einstiegspunkt in eine Suche mit Hipikat benutzt werden." Gib mir alle Informationen, die mit diesem Eintrag zu tun haben".

Das Resultat ist eine Liste, wie in Abbildung 5, wobei in dieser noch die Felder Relevance und Confidence fehlen (wird später erklärt). Aus Bequemlichkeitsgründen kann diese noch sortiert werden und es können Einträge herausgelöscht werden, was keinen Effekt auf den Server hat.

Abbildung 5

Interessiert sich der Benutzer nun für einen der Einträge, so öffnet sich der entsprechende Eintrag innerhalb von Eclipse. Z.B. öffnet sich eine Java Klasse im entsprechenden Editor. Andere Orte sind: CVS Repository View, Navigator, Bugzilla Editor...

4.1.6 Fallbeispiel: Eine Ãnderung an Eclipse

Um zu beweisen, das Hipikat seinen Nutzen hat, wurde ein Fallbeispiel gemacht.

Davor Cubranic, einer der Autoren Hipicats, bekam die Aufgabe den CVS Repository Browser um eine Funktion zu ergänzen und sich dabei auf Hipikat zu stützen.

Sobald im CVS Browser eine neue Version angelegt wird, musste die Ansicht manuell aufgefrischt werden, damit diese auftaucht. Diese Unannehmlichkeit zu beseitigen war die Aufgabe Davors.

Seine Erfahrungen mit Eclipse beschränkten sich auf das Editieren kleinerer Plugins, so dass er als Neuling im Kernbereich anzusehen war.

Eclipse eignet sich hervorragend als Beispielprojekt, da es sehr gross ist und eine reichlich gefüllte Bugzilladatenbank hat.

das Gruppengedächtnis auf dem Server wurde erzeugt und innerhalb Bugzillas wurde eine Aufgabe eingerichtet, deren ID an Davor gegeben wurde.

Die Suche mit Hipikat startet über diese ID.

Da die Aufgabe noch nicht abgearbeitet wurde, gibt es keine Verknüpfungen zu anderen Informationen innerhalb der Datenbank, ausser den rekursiven "similar to". Eine Anfrage lieferte zu dem Zeitpunkt nur ähnliche Bugzillaeinträge, die aufgrund von textueller Ähnlichkeit gefunden werden, aber eventuell schon behoben wurden und daher verknüpfte Elemente haben, aus denen man etwas lernen kann.

Einzelne Elemente dieser Resultatliste zeigen auf Bugzillaeinträge, die innerhalb von Eclipse in einer Bugzilla-View geöffnet werden können. An dieser Stelle kann dann eine weitere HipiKat Anfrage gestellt werden die dann auch andere Informationen, wie CVS Versionen, einzelne Klassen oder Dokumente enthält.

In Davors Fall wurde ein Eintrag gefunden, der schon behoben war und in der 2. Anfrage Klassen lieferte, die über die implements Verknüpfung gefunden wurden und eine ähnliche Funktionalität implementierten, wie die zu realisierende.

Die Aussagekraft dieser Untersuchung ist nicht aussagekräftig, da sie nur auf ein Projekt beschränkt war und sich sehr wenige Probanden daran beteiligten.

4.1.7 Genauigkeit

Im betrachteten Eclipse core Projekt existieren im Jahr 2002 ca. 800.000 Zeilen Code, 21.668 Bugzillaeinträge mit 72.536 Kommentaren, 125.429 CVS Revisionen, 36.864 Newsgroupeinträge und 1.459 Webseiten.

Von 9.418 abgearbeiteten Bugs wurden vom Hipikatserver durch den log- und activity matcher 5.688 (ca. 60 Prozent) Einträge untereinander in Beziehung gesetzt. Weitere 2.810 wurden verbunden, obwohl sie nicht geschlossen wurde. Von ihnen wird

angenommen, dass sie grösstenteils falsch sind. Sie haben einen sehr niegrigen Relevanzwert. Dieser Relevanzwert, oder auch Genauigkeit genannt, errechnet sich z.B. aus dem zeitlichen Abstand zwischen einem CVS Commit und der Ãnderung einer BugzillaID. Ist ein CVS Eintrag länger als 5 Minuten von einer Ãnderung entfernt, nimmt die Relevanz kontinuierlich ab.

Von den fälschlicherweise verbundenen 2.810 Einträgen haben beispielsweise 2/3 eine sehr geringe Relevanz.

4.1.8 Probleme und Zusammenfassung

Hipikat personalisiert den Benutzer nicht. Daher können keine Aussagen über Nützlichkeit von Hipikat-Resultaten gemacht werden. Es können genauso wenig Benutzerprofile angelegt werden, die bestimmte Vorlieben von Nutzern aufzeigen (z.B. zeige schon gesehene Codebereiche mit einer niedrigeren Relevanz, da diese sehr wahrscheinlich schon bekannt sind).

Hipikat ist nicht von einer bestimmten Formatierung von Dokumentationen im Code oder in BugzillaReports abhängig. Daher ist ein "Störrauschen" in den Resultaten wahrscheinlicher.

Dafür muss der Benutzer weder seine Programmiergewohnheiten ändern oder eine aufwendige Anfragesprache lernen. Die Genauigkeit wird jedoch stark erhöht, wenn der Benutzer Hipikats Funktionsweise kennt und BugzillaIds in CVS Commits, Newsgroupsbeiträgen und Webseiten einträgt, sowie Bugziallaeinträge sehr zeitnah nach dem implementierenden CVS Commit schliesst.

Die Einarbeitungszeit wird stark verringert, da sofort Ansatzpunkte geliefert werden, die zur Lösung einer Aufgabe in Bugzilla eventuell hilfreich sind.

Der Programmierer muss dafür im optimalen Fall nicht einmal wissen, wonach er sucht. Eine Hipikatanfrage kann immer formuliert werden als: "Gib mir Informationen, die Hilfreich sind zu folgender Klasse/BugzillaId...". Es kann jedoch auch sein, dass zu einer Anfrage keine relevanten oder falsche Daten vorhanden sind. Hier hilft der Relevanzwert weiter, der jedoch auch nicht immer korrekt sein muss.

Hipikats Berechnung des Relevanzwertes berücksichtigs nicht den Autor der Information. Ein Webdokument, das von Autor einers relevanten Bugzillaeintrages geschriebn wurde ist eventuell interessanter, als eins von einem beliebigen anderen geschriebenen.

Hipikats Zentrum bilden BugzillaIds, was Hipikat bei Projekten, die nicht alle Aufgaben hierüber verwalten nicht benutzbar macht. Diese zentrale Position wird am vereinfachten Datenbankschema (Abbildung 3) klar, in dem eine Verbindung zwischen Dokumenten und File resivionen komplett fehlt und nur über Change Tasks zu machen ist.

Der bugzillazentrische Ansatz macht Hipikat für (mittlere und kleine) Projekte unbrauchbar, falls diese kein genügend gefülltes Bugzillasystem haben.

4.2 Beispielcode-Abfragen mit Strathcona [Rei02]

Strathcona ist ein Werkzeug, das Programmierern, die ein bestimmtes Framework oder ein fremdes API benutzen, helfen soll, sich damit zurechtzufinden.

Die benötigten Informationen werden aus Beispielprojekten gesammelt.

Frameworks erlauben Programmierern Applikationen in sehr viel kürzerer Zeit zu schreiben. Ein Programmierer, der sich in einem ihm fremden Framework zurechtfinden soll sieht sich teilweise sehr komplexen Abläufen gegenübergestellt, die eingehalten werden müssen, um damit korrekt umzugehen. Oftmals existieren keine strukturierten Anleitungen, wie diese zu benutzen sind, oder es ist aus dem Kontext nicht sofort ersichtlich, was zu tun ist. Für manche Frameworks existiert nur die API Dokumentation.

Bei weiter verbreiteten APIs kommt einem der Umstand entgegen, dass diese in vielen Beispielprojekten bereits verwendet wurden, für die die Quellen offen liegen. Diese Projekte können vom Benutzer durchsucht werden, um vergleichbare Stellen zu finden, in denen das Framework bereits korrekt angewendet wurde, was nicht immer sehr einfach ist, da die einzige Möglichkeit des Suchens textuell ist.

Strathcoma übernimmt nun das Suchen von relevanten Codebeispielen aus mehreren Beispielprojekten. Dafür greift es auf einen Server zurück, der bereits mit entsprechenden Beispielprojekten gefüllt ist und diese analysiert hat.

Der Client besteht aus einem Eclipse Plugin, das während des Programmierens die bisher geschriebene Klasse untersucht und Informationen an den Server weitergibt, der vergleichbare Stellen der Beispielprogramme heraussucht und an den Benutzer als Liste zurückgibt.

4.2.1 Der Server

Dem Server können verschiedene Beispielprojekte, oder auch Quelltextschnipsel übergeben werden. Diese müssen compilierbar sein, aber es ist nicht notwendig, das diese lauffähig sind.

Die Klassen werden in einer relationalen Datenbank mit all ihren Informationen abgelegt. Hierbei wird auch die Struktur der Beziehungen untereinander beibehalten.

”Funktion a mit den Parametern x, y und dem Rückgabewert z gehört zu Klasse H mit der Superklasse I” wäre eine solche Information. Es werden auch Funktionskörper analysiert, also welche Typen instanziert und welche Funktionen aufgerufen werden.

Abbildung 6

4.2.2 Der Client

Eine Anfrage eines Clients an den Server bezieht sich immer auf : eine Klasse C, eine Methode m und/oder eine Variablendeklaration f.

Der Client analysiert den Quelltext, den ein Programmierer schreibt und extrahiert für (C,m,f):

- superklasse und interfaces von C

- Membervariablen von C
- Funktionsaufrufe aus m.

Hierbei stützt er sich auf den Eclipse Java Parser, der sehr Fehlertolerant ist.
Der gewonnene strukturelle Kontext wird an den Server geschickt.

4.2.3 Vergleichsheuristiken

Hat der Server ein AnfrageSet (C,m,f) mit allen Informationen übermittelt bekommen, so wendet er sechs Heuristiken zum Suchen von ähnlichen Passagen in den Beispielprojekten an.

Diese Heuristiken wurden so einfach wie möglich gehalten.

4.2.4 "Inheritance" Heuristik

Für diese Heuristik wird angenommen, das der Programmierer den Teil des zu benutzenden Frameworks bereits kennt und daher seine grade geschriebene Klasse bereits von den richtigen Klassen erbt, Interfaces implementiert und eventuell ähnliche Membervariablen deklariert hat.

Daher wird in der Datenbank nach Klassen gesucht, die diese Ähnlichkeiten aufweisen. Die Liste wird nach Anzahl der Ãberdeckten Vererbungen geordnet.

4.2.5 "Calls" Heuristik

Diese drei Heuristiken stützen sich auf die Funktionsaufrufe einer Methode m.

Als Szenario kann der Programmierer gesehen werden, der einen Ablauf grösstenteils verstanden hat, dem aber "die letzten Meter" fehlen (z.B. fehlt in einer Befehltsfolge zur Initialisierung der letzte Schritt).

"Basic Calls" Heuristik: In der Datenbank enthaltene Funktionen, die die angegebenen Aufrufe enthalten werden weitergegeben. Die Liste wird nach grösstmöglicher Ãberdeckung sortiert.

Hierbei werden jedoch auch sehr lange Methoden weitergereicht, die die entsprechenden Aufrufe zwar enthalten, deren Kontext aber nicht mehr stimmt. Hierbei wird das Ergebnis durch die grosse Menge an Aufrufen im Vergleich zu den wenigen in m enthaltenen verfälscht.

Die "Calls Best Fit" Heuristik liefert nur Ergebnisse, bei denen das Verhältnis von enthaltenen zu gesamnte Aufrufen den Faktor 0.4 nicht überschreitet. Dieser Faktor ist aus Erfahrungswerten entstanden und muss nicht auf jedes Projekt anwendbar sein.

Die "Calls with Inheritance" Heuristik hat als Zusicherung zur "Calls Best Fit", dass mindestens ein Interface/Superklasse gemein sein muss.

4.2.6 "Uses" Heuristik

Die "basic uses" Heuristik betrachtet alle verwendeten Typen innerhalb einer Methode (inklusive der übergebenen Parameter und Rückgabewerte) und sammelt Methoden geordnet nach grösstmögliche Ãberdeckung. Dies ist nützlich, wenn ein Entwickler schon weiss, welche Klassen bestimmte Funktionen enthalten, oder er diese Klassen benutzt,nur nicht richtig.

Die "uses with inheritance" Heuristik hat als weitere Zusicherung, die Ãberdeckung mindestens einer vererbenden Klasse/Interface.

4.2.7 Antwort auf eine Anfrage

Sobald der Server eine Anfrage erhalten hat und die Heuristiken ihre Antwortlisten erstellt haben werden die zehn besten Ergebnisse ausgewählt. Diese errechnen sich aus der Häufigkeit, mit der eine bestimmte Klasse von den Heuristiken als relevant eingestuft wurde.

Von jedem dieser zehn Ergebnisse werden die folgenden drei Informationen extrahiert und dem Benutzer zur Verfügung gestellt (Abbildung 6):

- Eine in vereinfachter UML Darstellung vorliegende strukturelle Darstellung der Ähnlichkeiten zwischen gefundener und angefragter Klasse

- Eine Liste der Heuristiken, die diese Klasse als relevant einstufen und der Grund dafür (welche Klasse,Methode,Varliable wird überdeckt)

- Der Quelltext der Klasse selbst

Abbildung 7

4.2.8 Testumgebungen

Strathcona wurde unter verschiedenen Bedingungen getestet.

Als Framework wurde Eclipse gewählt, das ein Framework für Plugin Programmierer bietet. Aus der Eclipse Community sind hunderte von Plugins für die verschiedensten Aufgaben entsprungen, die alle als Beispielprojekte verwendet werden könen.

An zwei Probanden wurden vier unterschiedliche Aufgaben gestellt, die von diesen gelöst werden sollten. Die Beiden hatten wenige Monate Erfahrung mit Eclipse, waren jedoch mit Java vertraut. Zur Lösung der Aufgabe standen nur die Eclipse eigenen Werkzeuge zur Verfügung (API Browser...) sowie das Strathcona Plugin.

Abbildung 7 zeigt eine Statistik der Testumgebung. Alle vier Aufgaben wurden erfolgreich abgeschlossen. In fünf von acht Fällen haben die Entwickler die Benutzung von Strathcona als nützlich empfunden[Rei02].

Die Aussagekraft dieser Untersuchung ist nicht aussagekräftig, da sie nur auf ein Projekt beschränkt war und sich sehr wenige Probanden daran beteiligten.

4.2.9 Probleme und Zusammenfassung

Der erfolgreiche Einsatz von Strathcona hängt davon ab, wie gut die Beispielprojekte sind, mit denen die Datenbank des Servers gefüllt wird.

Selbst wenn ein Programmierer auf dem richtigen Weg ist in der Benutzung eines Frameworks ist nicht gesagt, das ein Beispiel, das eine relevante Klasse, die von Strathcona geliefert wird auch wirklich relevant ist. Es könnte sein, dass das Framework in dieser in einem anderen Kontext benutzt wird. In dieser Situation ist der Benutzer darauf angewiesen, dass der gelieferte Code z.B. kommentiert ist.

Ein weiteres Problem stellt "Copy and Paste" Programmieren dar. Das Ãbernehmen von Codepassagen aus den gelieferten Beispielen kann dazu führen, dass unerwünschte Nebeneffekte auftreten, die daraus resultieren, dass das Ãbernommene nicht verstanden wurde und überflüssige Passagen enthält.

Die letzten zwei Probleme werden darurch abgemildert, das immer mehrere Resultate geliefert werden, die verglichen werden können.

Die Testumgebung hat gezeigt, dass Strathcona gültige Ergebnisse liefern kann, aber seine Probleme und Beschränkungen mit sich bringt.

Die Hauptaufgabe des Werkzeuges liegt jedoch darin, dem Entwickler eines Frameworks eine grosse Last von den Schultern zu nehmen: die Dokumentation.

Für die Dokumentation eines Frameworks ist eines besonders wichtig: gute Beispiele, die veranschaulichen, wie die Abläufe eines Frameworks funktioneren.

Dies kann über eine strukturierte Sammlung von gut kommentiertem Beispielcode auf einer Homepage geschehen. Strathcona übernimmt diese Aufgabe, indem es seinen Benutzern über ein Eclipse Plugin die Möglichkeit gibt eine Anfrage zu senden. Diese Anfrage benötigt keine eigene Anfragesprache, um gestellt zu werden, sondern wird aus dem aktuellen Stand des Quelltextes der bearbeitenden Klasse errechnet.

4.3 andere Werkzeuge

Neben Hipikat und Strathcona existieren viele weitere "light weight knowledge sharing tools" die auf verschiedenen Informationsquellen aufbauen. Das einzig weitere Tool, das hier kurz vorgestellt wird ist ROSE.

4.3.1 ROSE "Mining Software Histories to Guide Software Changes" [Tho04b]

Ein weiteres, sehr interessantes Gebiet des "Light Weight Knowledge Sharing" ist das CVS zentrische. Hierbei werden Informationen nur aus dem Versionssystem gewonnen und aufbereitet.

Das Werkzeug ROSE [Tho04b] beispielsweiswe geht davon aus, dass bestimmte Vorgänge in einer Applikation sich immer wieder wiederholen.

So wie Amazon die Funktion "Kunden, die dieses Buch gekauft haben, haben auch folgende gekauft..." hat, könnte in Softwareprojekten eine Funktion angeboten werden,

die ausformuliert lauten könnte: ”Programmierer, die diese Klasse geändert haben, haben auch folgende geändert...”.

Hierbei wird versucht anhand der Informationen aus dem CVS vorauszusagen, welche Teile des Quellcodes ein Programmierer als nächstes bearbeiten muss.

Werden z.B. an einer Methode Änderungen gemacht, so werden ähnliche CVS Transaktionen gesucht. Von einer CVS Transaktion (einem CVS Commit) wird angenommen, das alle ihre enthaltenen geänderten Klassen semantisch eine Einheit bilden. Wenn Beispielsweise in 90 Prozent der Fälle, in denen die Methode foo der Klasse Bar geändert wurde auch eine Konstante zur Klasse Konstants hinzugefügt wurde, so wird ROSE auf dieses hingewiesen.

5 Zusammenfassung

Die Faktoren in verteilten FOSS Projekten lassen einen traditionellen Weg für Wissensmanagement kaum zu. Ein Transfer von Wissen über synchrone Informationsmedien ist zwar sehr effektiv, aber auch aufwendig und muss jedes mal repliziert werden. Durch diesen Umstand werden in FOSS Projekten vor allem synchrone Informationsmedien gepflegt.

Diesen Wissensspeichern haben eines gemein: Das Akzeptanzproblem.

Würde jedes Projekt eine gut ausgewählte Anzahl an Dokumentationswerkzeugen benutzen und in ihnen immer qualitativ hochwertige Dokumentationen schreiben, die keine Fragen offen liessen, so würden "light weight"-Werkzeuge überflüssig sein. Die hier vorgestellten Werkzeuge versuchen alle eine bestimmte Lücke auszufüllen: fehlende Dokumentation. Sie haben ihre Stärken und Schwächen, die sie jeweils für bestimmte Umgebungsfaktoren und/oder Einsatzgebiete nützlich machen. Passen diese nicht, so kehrt sich der Effekt ins gegenteilige um und die Benutzer werden eher auf falsche Wege geschickt, als das ihnen geholfen wird.

Keines von ihnen bietet einen zuverlässigen Ersatz für traditionelle Dokumentation. Sie können jedoch unter bestimmten Umständen eine grosse Hilfe sein.

References

[???a] ??? Herausforderung wissensmanagement.

[???b] ??? Learning communities und wissensmanagement.

[???c] ??? Wissensmanagement.

[???d] ??? Wissensmanagement: Management von expertise.

[Dav] Davor Cubranic, Gail C. Murphy. Hipikat: Recommenting pertinent software development artifacts.

[Dav03] Davor Cubranic, Reid Holmes. Tools for light weight knowledge sharing in open-source software development. 2003.

[Kev03] Kevin Crowston, Hala Annabi. Effective work practices for software engineering: Free/libre open source software development. 2003.

[Rei02] Reid Holmes, Gail C. Murphy. Using structural context to recomment source code examples. 2002.

[Tho] Thomas Chau, Frank Maurer. Tool support for inter-team learning in agile software organisations.

[Tho04a] Thomas Chau, Frank Maurer. Integrated process support and light weight knowledge sharing for agile software organisations. 2004.

[Tho04b] Thomas Zimmermann, Peter Wei?gerber. Mining version histries to guide software changes. 2004.